Listas Lineares

Centro Federal de Educação Tecnológica Celso Suckow da Fonseca
CEFET-RJ

Listas Lineares

- **Coleção de elementos do mesmo tipo dispostos linearmente que podem ou não seguir determinada organização.**
- **Número de elementos (nós) de uma lista é chamado de comprimento ou tamanho da lista.**
- **Exemplos:**
  - **Lista de chamadas**
  - **Lista telefônica**
  - **Lista de compras**
  - **Lista de convidados**

Listas Lineares

- **Onde armazenar?**
  - **Numa estrutura de dados estática (arranjos) ou numa estrutura dinâmica de dados (ponteiros).**
    - **Cada uma possui vantagens e desvantagens.**
- **Principal propriedade estrutural:**
  - **A posição relativa dos elementos são dispostos linearmente.**
- **Propriedades:**
  - **Se $n = 0$, diz-se que a lista está vazia; caso contrário, são**
  - **x_1 é o primeiro elemento da lista;**
  - **x_n é o último elemento da lista;**
  - **$\forall$ i, 1 < i < n, o elemento x_i é precedido por x_{i-1} e seguido de x_{i+1}**

Operações de uma lista

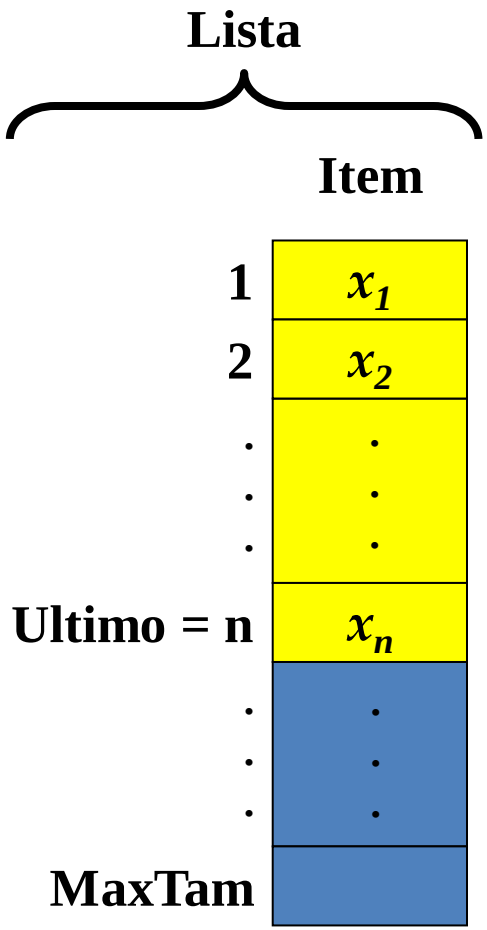
1. **Criação de uma lista linear vazia.**
2. **Acesso ao i-ésimo elemento de uma lista para consultá-lo ou alterá-lo.**
3. **Inserção de um novo elemento antes (ou depois) do i-ésimo elemento.**
4. **Remoção do i-ésimo elemento.**
5. **Cópia de uma lista.**
6. **Exclusão de uma lista.**
7. **Combinação (intercalação) de duas ou mais listas em uma única lista.**
8. **Particionamento de uma lista em duas ou mais.**
9. **Determinação do tamanho.**
10. **Ordenação dos elementos.**

Exemplos de Funções

- **FazListaVazia(Lista)**
  - **Faz a lista ficar vazia.**
- **Insere(x, Lista)**
  - **Insere *x* após o último item da lista.**
- **Retira(p, Lista, x)**
  - **Retira o item *x* que está na posição *p* da lista, retirando-o da lista e deslocando os itens a partir da posição *p+1* para as posições anteriores.**
- **ListaVazia(Lista)**
  - **Esta função retorna *verdadeiro* se a lista está vazia; caso contrário, retorna *falso*.**
- **Imprime(Lista)**
  - **Imprime os itens da lista na ordem de ocorrência.**

Listas Lineares usando arranjo

- **Os itens da lista são armazenados em posições *contíguas* de memória.**
- **A lista pode ser percorrida em qualquer direção.**
- **A inserção de um novo item pode ser realizada após o último item com custo constante.**
  - **Se a inserção for no meio da lista, então deverá ser realizado um deslocamento de todos os itens localizados após o ponto de inserção.**
- **Retirar um item do início da lista requer um deslocamento de itens para preencher o espaço deixado vazio.**

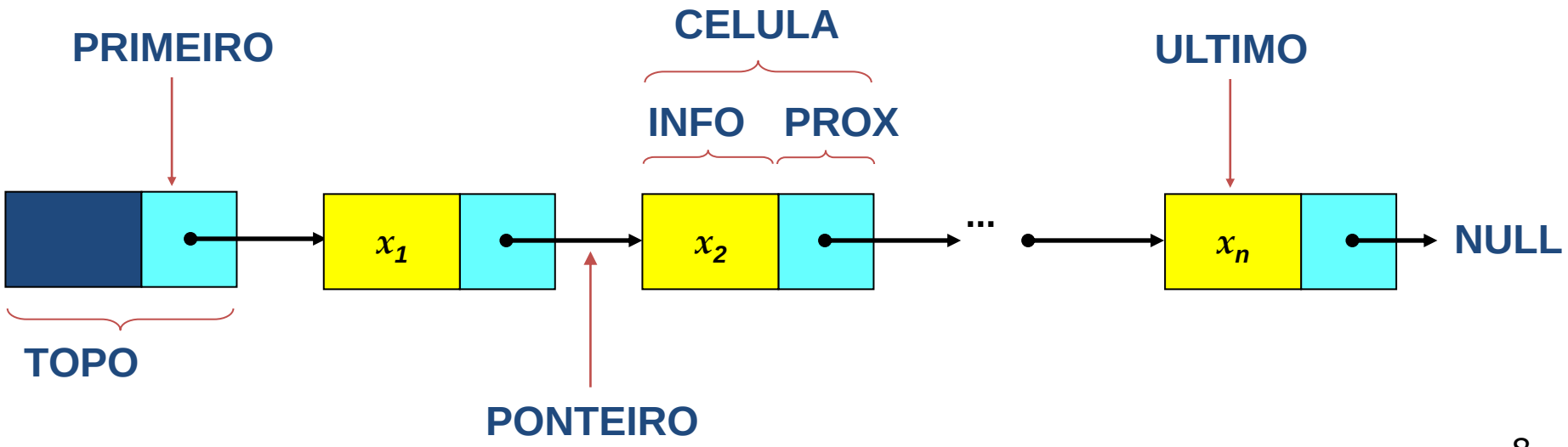


Listas Lineares usando arranjo

- **Vantagem**
  - **acesso indexado**
  - **economia de memória**
    - **ponteiros, ou índices, são implícitos nesta estrutura.**
- **Desvantagem**
  - **o custo para inserir ou retirar itens da lista, pode causar um deslocamento de todos os itens, no pior caso;**
  - **em aplicações em que não existe previsão sobre o crescimento da lista, a utilização de arranjos em determinadas linguagens pode ser problemática**
    - **o tamanho máximo da lista tem que ser definido em tempo de compilação.**

Listas Lineares usando ponteiros

- **Em uma implementação de listas usando ponteiros, cada item da lista é encadeado com o seguinte através de uma variável do tipo *Ponteiro*.**
  - **permite utilizar posições não contíguas de memória, sendo possível inserir e retirar elementos sem haver necessidade de deslocar os itens seguintes da lista.**



Listas Lineares usando ponteiros

- **Vantagem**
  - **permite inserir ou retirar itens do meio da lista a um custo constante, aspecto importante quando a lista tem que ser mantida em ordem.**
  - **em aplicações em que não existe previsão sobre o crescimento da lista , pois neste caso o tamanho máximo da lista não precisa ser definido a priori.**
- **Desvantagem**
  - **utilização de memória extra para armazenar os Ponteiros.**

Estruturas Auto-referenciadas

- **Uma estrutura auto-referenciada (recursiva) contém um membro ponteiro que aponta para uma estrutura do mesmo tipo.**

```
struct no {
    int info;
    struct no *prox;
};
 typedef struct no  *noPtr;
 noPtr lista;
```

Estruturas Auto-referenciadas

- **Definiu-se uma estrutura com dois membros:**
  - **um inteiro info e um membro ponteiro denominado prox.**
  - **o membro prox aponta para uma estrutura do mesmo tipo que o da declarada ⇨ estruturada auto-referenciada.**
  - **o membro prox é chamado *link* (ligação)– ou seja, prox pode ser usado para "ligar" uma estrutura com outra do mesmo tipo.**

As estruturas auto-referenciadas podem ser ligadas entre si para formar estruturas do mesmo tipo.

Usadas para criar listas, filas, pilhas e árvores utilizando alocação dinâmica de memória.

Alocação Dinâmica de Memória

- **Criar e manter estruturas dinâmicas de dados exige *alocação dinâmica de memória***
  - **capacidade de um programa obter, em tempo de execução, mais espaço de memória para conter novos nós e liberar espaço não mais necessário.**
  - **O limite máximo da alocação dinâmica de memória pode ser a quantidade de memória física disponível no computador ou a quantidade de memória virtual disponível em um sistema de memória virtual. Normalmente estes limites são menores, pois a memória disponível deve ser compartilhada por muitos usuários.**

Alocação Dinâmica de Memória

- **Em C usamos as funções:**
  - ***malloc***
  - ***free***
  - **operador *sizeof***
- **A função *malloc* utiliza como parâmetro o número de bytes a serem alocados e retorna um ponteiro para void para a memória alocada. A função malloc é usada normalmente com o operador sizeof.**

  prox = malloc(sizeof(struct no));

Alocação Dinâmica de Memória

- **sizeof(struct no) determina o tamanho em bytes de uma estrutura do tipo struct no, aloca uma nova área de sizeof(struct no) na memória e armazena na variável prox um ponteiro para a memória alocada. Se não houver memória disponível, malloc retorna um ponteiro NULL.**
- **A função free libera memória alocada**
  - **a memória é retornada ao sistema para que possa ser realocada no futuro.**
  - **Para liberar memória alocada dinamicamente pela chamada precedente a malloc, use a instrução free(no);**